AN3109
Application note

Communication peripheral FIFO emulation with DMA and DMA timeout in STM32F10x microcontrollers

Introduction

The STM32™ communication peripherals have a single transmit buffer and a single receive buffer. The user software should therefore retrieve data from the receive buffer before the data are overwritten by the next received data. With interrupts, there is a risk of data overflow. The STM32's DMA feature prevents data overflow but, usually, the number of data items to be received is not known in advance, and it is variable (from one reception sequence to the next). Consequently, in reception, the end of transfer cannot be detected.

The solution is to implement an emulated FIFO based on both DMA and interrupts, a DMA timeout is required to indicated to the application that no further data will be received.

The only requirements for FIFO implementation are that the original data and their order are preserved. Because it is so simple, FIFO structures are easily implemented in both hardware and software.

This application note is based on the implementation of a simple 200-byte circular buffer, but the principle can be extended to buffers of any size. Likewise, the peripheral used here is the USART but the same principle can be adopted for any other communication peripheral.

The aim of this document is to show how to build an efficient circular FIFO using the STM32F10x's DMA, and to provide methods for the implementation of DMA timeout.

This application note is organized into two parts. It first gives a FIFO overview: it discusses FIFO emulation in the STM32's system RAM and provides a description of the software required for FIFO implementation. Then it provides two methods for the implementation of DMA timeout.

This application note assumes that the reader is familiar with the STM32's DMA as described in the STM32F10xx reference manual, RM0008, available for the STMicroelectronics website *www.st.com*.

Contents

List of figures

1 FIFO emulation with DMA

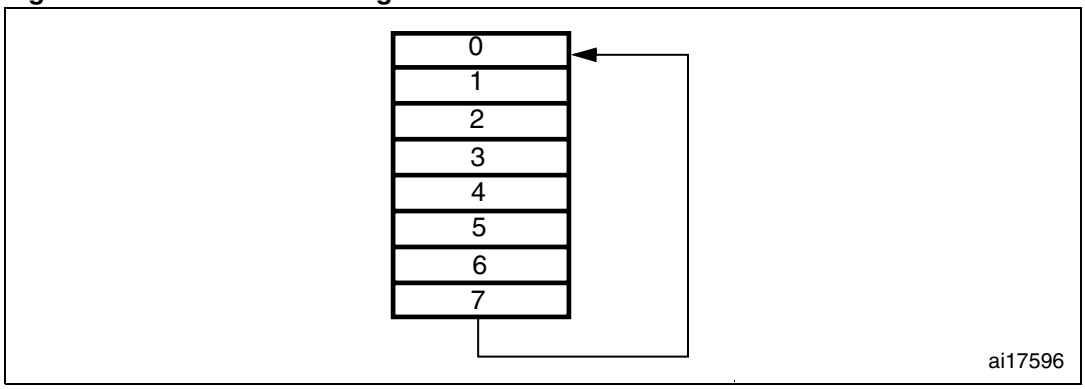
1.1 FIFO overview

FIFO is an acronym for first in, first out. It is an abstraction of the ways of organizing and manipulating data in terms of time and order. This expression describes a queue-processing technique. It consists in the servicing of conflicting demands based on the principle of first-come, first-served: the data that come in first are handled first, the data that come in next are served after the first data in are removed, etc.

FIFO (first in first out) requires that the first data item input is the first output. It is necessary to keep track of the amount of data items in the buffer so that data are not dropped or duplicated.

A circular buffer is an efficient way of implementing a FIFO. *Figure 1* shows an 8-byte circular buffer.

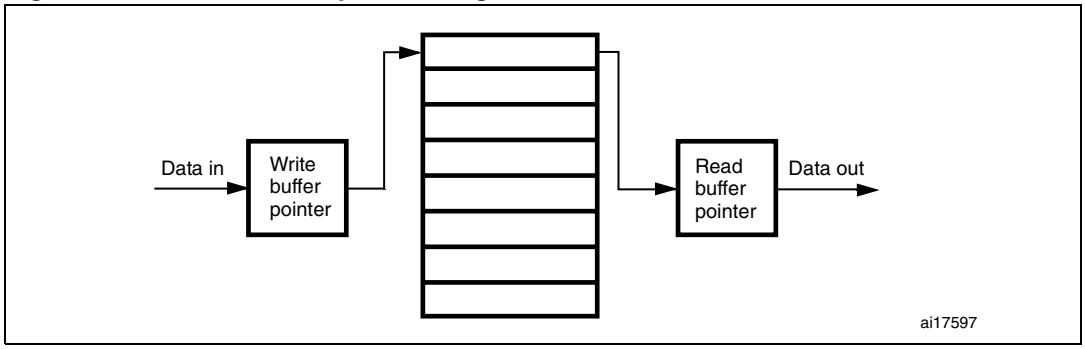
Figure 1. Circular buffer diagram



This buffer has 8 bytes of storage. It is referred to as circular because the next position accessed after position 7 is position 0.

Figure 2 shows two pointers used to control the movement of data into and out of the buffer.

Figure 2. Circular buffer pointer diagram



The write pointer points to the next available buffer location to be written. It is incremented when data are placed into the buffer. The read pointer points to the next buffer location to be read. It is incremented when data are fetched from the buffer.

1.2 RAM FIFO emulation in STM32 microcontrollers

The DMA capability of STM32 microcontrollers greatly simplifies the FIFO implementation. DMA is an efficient way of implementing RAM FIFO based on the principle described in *Section 1.1: FIFO overview*.

DMA features that simplify the FIFO implementation:

- Independent source and destination transfer sizes (byte, half-word, word), to emulate packing and unpacking
- Support for circular buffer management
- Access to Flash memory, SRAM, APB1, APB2 and AHB peripherals as source and destination
- Programmable number of data items to be transferred: up to 65536

All these features concur to minimize the software overhead associated with data storage. The DMA memory increment mode is very useful because, with it, the data pointer can be automatically incremented.

Here, the DMA buffer emulates the FIFO buffer. The write buffer pointer (DMA pointer) is automatically incremented and the DMA count is automatically decremented when the FIFO is filled.

The read buffer locations are incremented by software every time data are retrieved from the FIFO buffer.

1.3 FIFO software implementation

1.3.1 Implementation

This example describes the basic architecture of a circular buffer FIFO for a communication peripheral. The USART is provided as an example.

Data are received and stored into the DMA circular buffer, where they remain until they are removed and manipulated.

The transmitter transmits n data items using DMA. The message length (n) is known in advance.

The receiver receives m (potentially unknown) data items using DMA. It is still possible to get the peripheral's RXNE interrupt even when using DMA. In fact, the interrupt from the peripheral emulates the FIFO nonempty interrupt.

For reception:

- There is no need to clear the RXNE flag in the receive interrupt routine, as it was automatically cleared by the DMA data read operation. However, the interrupt remains pending in the NVIC (even though the RXNE flag is no longer set).
- Two buffers are used:
  - RxBuffer2: it is defined as the DMA memory base address from which data will be read. This buffer emulates the FIFO buffer.
  - RxBuffer2_SW: it is a software buffer used, inside the receive interrupt routine, to transfer the received data from the FIFO. It is the final data storage destination.

- Inside the receive interrupt routine, the RAM address pointer/count of the DMA are used to indicate:
  - how many data bytes are available in the FIFO buffer (RxBuffer2) to be transferred to the final data storage buffer (RxBuffer2_SW)
  - which is the current FIFO location of the data

Incoming data are temporarily stored into the FIFO buffer, when the receive DMA requests are serviced. Data retrieval from the FIFO and/or processing is/are triggered by the receive interrupts.

In the firmware example provided with this application note, USART1 or USART3 (depending on the STMicroelectronics evaluation board used) serves as the transmitter. USART2 serves as the receiver using DMA and interrupts. The transmitter (USART1 or USART3) transmits 250 data bytes to USART2.

The receive DMA buffer length is equal to 200 and is defined as circular.

When the USART2 receive interrupt is triggered, the FIFO is read and the RxBuffer2_SW buffer is filled based on the current DMA pointer/count.

1.3.2 Advantages

DMA is an efficient way of implementing a configurable FIFO for the different communication peripherals.

This emulated FIFO implementation is needed in reception using DMA when the data length is not known in advance. There are two major cases:

- A continuous flow of data is received and the incoming data flow can be processed by the application as soon as it is received. This case could be directly implemented using the USART receive interrupt. However, FIFO emulation has the major advantage of reducing the real-time/latency requirements on the USART interrupt. Once the data are present in the FIFO buffer, they can be processed by software when the CPU is free of other higher-priority tasks.
- The end of the data block can be determined by the data content —e.g. the software may check for the presence of an SOF character in the received flow. Another way consists in detecting the end of the data block through a pause (long interval without any data reception) in the data flow. This particular case is discussed in *Section 1.3: FIFO software implementation*.

2 Receive DMA timeout

2.1 Overview

When using DMA for reception, the end of transfer cannot be detected if the number of data items to be received is not known in advance.

Consequently, a DMA timeout has to be implemented when no further data are received over a certain period.

In this section, DMA timeout means that an interrupt is generated if no data have been received for a certain, user-defined period.

2.2 DMA timeout methods

For the implementation of timeout, two methods are described.

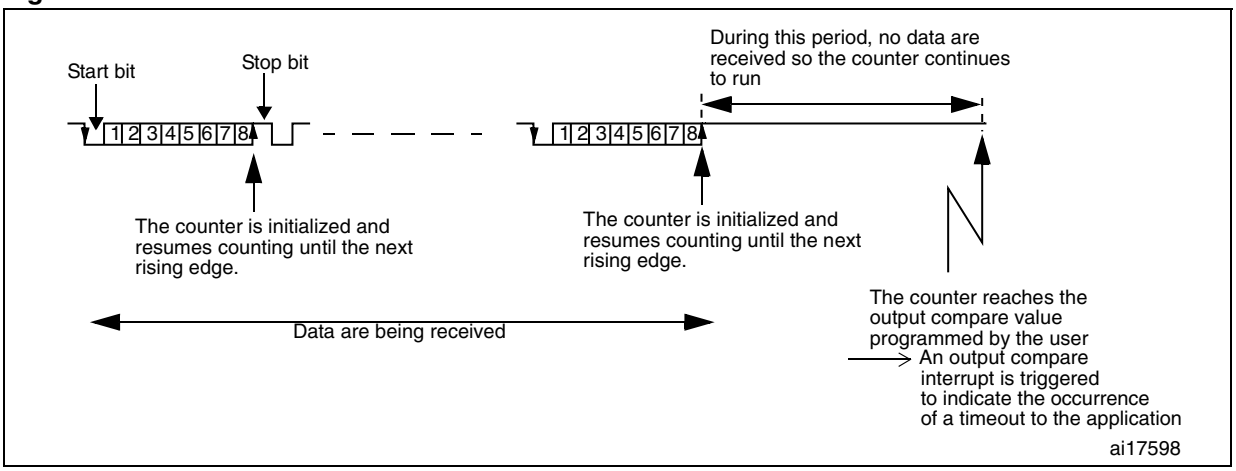
2.2.1 Method1: Connecting USART_RX to a timer input capture

The idea is to use a timer in slave reset mode, whose counter is reinitialized in response to rising edges on an input capture connected to the USART receive pin (USART_RX). On each rising edge of the receive pin, the timer counter is reset.

By programming the output compare value with the desired timeout, when no data are received during this period (no rising edge of USART_RX), the counter continues its operation until it reaches the output compare value corresponding to the user-defined timeout. Consequently, the timer generates an output compare interrupt (already enabled) that informs the application of the occurrence of a timeout.

Figure 3 illustrates the first method.

Figure 3. Method 1



2.2.2 Method2: Using the system timer and USART receive interrupts

This method is used to detect a timeout in a range of 1 to 2 system timer periods/ticks.

The idea is:

- to enable USART receive interrupts and receive DMA requests
- to enable the system timer overflow interrupt
- when the USART receive interrupt is triggered:
  - a variable is set
  - the USART receive interrupt is disabled

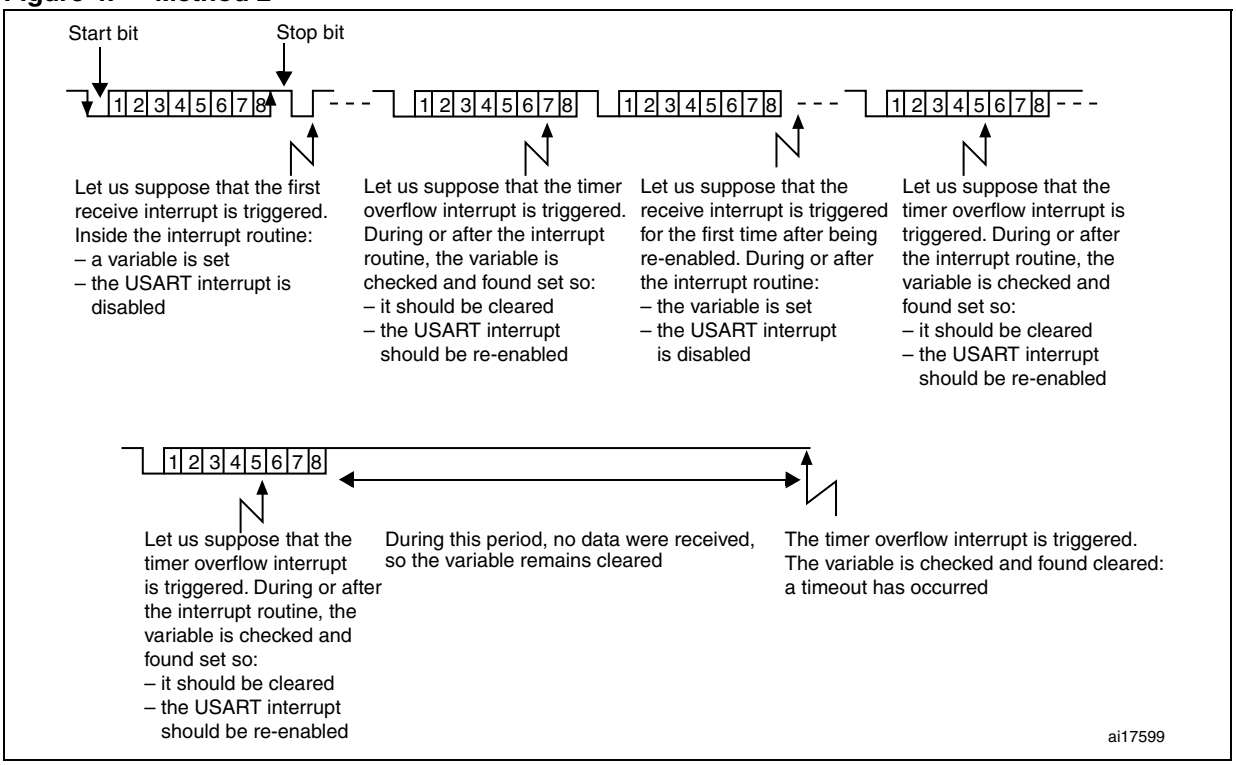
In this way, we ensure that at least one reception operation occurs.

- When the timer interrupt overflow is triggered:

  Check the variable value:
  - If it is set, this means that no timeout occurred. You have to clear the variable and re-enable the USART receive interrupts.
  - If it is cleared, this means that a timeout occurred (no USART receive interrupt occurred during the programmed period).

Figure 4 illustrates the second method.

Figure 4. Method 2



Note: This method is not implemented. Only the method described in the Section 2.2.1 is implemented.

3 Revision history

Table 1. Document revision history

Date	Revision	Changes
14-Dec-2009	1	Initial release.